

CGN User Manual

Felip Alàez Nadal

March 22, 2008

Contents

1	Introduction	4
1.1	Asking and contributing to CGN	4
1.2	No warranty	4
2	Installing CGN	4
2.1	Which gnuplot version to use?	5
2.2	Using CGN on Windows	5
2.2.1	Limitations of CGN on Windows	6
2.3	Where does CGN work?	6
3	Starting and closing gnuplot	7
4	The Gnuplot API	7
4.1	The plot macro	7
4.2	The splot macro	8
5	The CGN API	8
5.1	plot2d	8
5.2	plot3d	9
6	Using multiple connections	9
6.1	The with-gnuplot-connection macro	9
7	The easy-plotter utility	10
8	Plotting functions	10
8.1	Surfaces	17
9	Plotting scatter graphics	17
9.1	The simple way	18
9.2	plot-points-and-function	19
10	Ploting histograms	21
11	Configuring gnuplot	22
12	Printing	22
13	Saving/loading	22
14	Creating animations	23

15 Sending commands directly to gnuplot	23
16 Debugging CGN	24
17 Inside CGN	24
17.1 Extending CGN	25
18 Seeing release info	26
19 Changelog	26
20 Bibliography	28

1 Introduction

CGN is a library to control gnuplot from Lisp. Initially thought as simply a pipe to communicate with gnuplot, It has grown to provide syntactic sugar for the most common operations on Gnuplot: plotting graphics, printing, saving/loading...

CGN has become big enough to need a manual for itself. It comes here. I hope It to be useful. You can also see the examples at CGN's web at <http://common-lisp.net/project/nixies>

1.1 Asking and contributing to CGN

There are 2 mailing lists for CGN at <http://common-lisp.net/cgi-bin/mailman/listinfo/nixies-devel> and <http://common-lisp.net/cgi-bin/mailman/listinfo/nixies-announce>. You can also mail me at superratoli@gmail.com.

Contributions to CGN are welcome. Just send me them and I'll add them to CGN.

1.2 No warranty

CGN Copyright (C) Felip Alàez Nadal 2006-2008

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Steet, Fifth Floor, Boston, MA 02111-1307 USA

2 Installing CGN

CGN is easy to install. Simply start a REPL and write `(asdf-install:install 'CGN)`. This will download and install CGN for you. Note that CGN needs `ltk` to be installed. If you don't have `ltk`, this will download and install `ltk` too.

If you don't have a working internet connection available, download the source code manually, uncompress It and go to the directory that has appeared. Then start a REPL and write (asdf:oos 'asdf:load-op 'CGN). This will compile and load CGN. If you don't have ltk installed, you'll get an error. Please go to <http://cliki.net/ltk> and download ltk.

2.1 Which gnuplot version to use?

CGN works well on gnuplot newer than 4.2.0. You shouldn't use an older CGN, some things won't work. Histogram styles were introduced with gnuplot 4.2.0¹. The with-gnuplot macro doesn't work with gnuplot older than 4.0.0. CGN can be made to work on older gnuplots, but I encourage you to upgrade your gnuplot before trying to do that. It can be difficult and non portable.

2.2 Using CGN on Windows

CGN works on Windows, but you need to know how. There are two options to make It work:

1. When you use the with-gnuplot macro or the start-gnuplot function, use the :path parameter to tell CGN where gnuplot is installed.
2. Add gnuplot to the path.

Usually people doesn't know how to do the second. Well, if you want to add gnuplot to the path on a Windows machine ², first of all, you need to install gnuplot, of course. Then, add the directory where you installed It to the path. You can do It by clicking the left button on "My computer", then "Advanced Options" and "Environment variables". Modify the path variable to add the directory. E.g. : if you installed gnuplot at

```
c:\gnuplot
```

, your path variable must be

```
"c:\program files [more things] ; c:\gnuplot"
```

Then, go to the directory where gnuplot is installed and rename pgnuplot.exe to gnuplot.exe. That's all! ³

¹This has been pointed by Dan Becker. Thanks you!

²This is though for Windows XP. Under other Windows versions this could change

³Yes! I've copied this from the CGN page at cliki.net

2.2.1 Limitations of CGN on Windows

The Windows version of gnuplot doesn't use the `-persist` parameter at all. Due to that, the `with-gnuplot` macro works a little different than in Linux⁴. You can use the `with-gnuplot` macro, but when it will show all the plots, it will wait for you to write something and push `intro` to return. This is the only change.

2.3 Where does CGN work?

CGN is very portable due to its use of `ltk`.

There are some tests that document and show some of the features of CGN. You can run them by writing `(cgn::plot-test)`, `(cgn::splot-test)`, `(cgn::plot2d-test)` and `(cgn::multiple-connection-test)`. A combination of some Lisp implementation and operative system is considered to work when it can run without trouble all these tests. At the moment, this includes:

1. CMUCL 19e on Suse Linux 10 Enterprise Edition for x86.
2. SBCL 1.0.14 on Suse Linux 10 Enterprise Edition for x86.
3. LispWorks 4.4 Personal Edition on Suse Linux 10 Enterprise Edition for x86.
4. CLISP 2.44 on Suse Linux 10 Enterprise Edition for x86.
5. SBCL 1.0.14 on OpenSuse 10.1 for x86_64.
6. CLISP 2.44 on OpenSuse 10.1 for x86_64.
7. Closure Common Lisp (aka OpenMCL) on OpenSuse 10.1 for x86_64.
8. Mac OS X.
9. CLISP 2.44 on Windows XP (32 bits)⁵.
10. LispWorks 4.4 Personal Edition on Windows XP (32 bits).

Older versions of CGN were tested on other combinations and were known to work, but nowadays we don't have such operative systems to test it. This includes OpenBSD (`sbcl`) and FreeBSD (`clisp`). Please, send us the operative system/ Lisp implementation that you use.

⁴CGN will try to guess which operative system is running using the `which-os` function.

⁵On Windows, `with-gnuplot` works a little different due to gnuplot not understanding `-persist`.

3 Starting and closing gnuplot

To use CGN, first of all you must start a gnuplot connection. When you have finished using It, you must close the conection. You can choose how to do that. Note that CGN only can only have 1 running connection.

start-gnuplot &key persistent (path "gnuplot") This function will start a gnuplot connection. It takes two keyword paramethers: *path* is the path to gnuplot. Use It if your gnuplot executable is not at the path of your os. *persistent* must be set to t is you want the gnuplot window to not close after closing the connection.

close-gnuplot This will close the gnuplot connection for you.

with-gnuplot (&key (os os) (path "gnuplot")) &body body
This is a macro and will need some explanation. *os* is a parameter which can be set to some symbol identifying the underlying operative system. Until CGN008, this had to be 'w32 is you were under a Windows pc, because some functions have to generate diferent code under Windows. Other values simply where treated as "non 'w32". CGN009 tries to guess which operative system is running. Due to that, you don't have to use this parameter if everything goes ok, but It still exists if you need. The path parameter has the same meaning that in the start-gnuplot function. Body is the lisp code that is gonna be executed when the connection is open.

This macro calls to start-gnuplot , executes body and then calls to close-gnuplot inside an unwind-protect construction. This way, we can be sure that the connection is being closed.

4 The Gnuplot API

4.1 The plot macro

The plot macro lispifies the syntax of the gnuplot command "plot something [options]" . Its prototype is

plot something-to-plot &rest options something-to-plot must be a string. options are symbols or strings, the same you would use in gnuplot. something-to-plot must be enclosed within " if It's a file name.

```
(with-gnuplot ()  
  (plot "sin(x)" with lines )  
)  
  
(with-gnuplot ()  
  (plot " 'data.txt' " title "MyData" )  
)
```

4.2 The splot macro

This macro is very similar to plot. Simply uses the splot command from gnuplot instead of plot.

5 The CGN API

CGN implements an API similar to that from Maxima. Usually, if you have a Maxima command to plot something, simply change [] by () and delete all the commas and you'll get something that works with CGN.

They all understand the following options

- connection This is an option that does not exist in Maxima. Use ('connection name) and CGN will know which active gnuplot connection to use for that plot.
- y
- legend
- xlabel This is the label to put at the x axis. ('xlabel "my-x-label")
- ylabel This is the label to put at the y axis. ('ylabel "my-y-label")
- logx
- logy
- box
- gnuplot_term_options
- gnuplot_out_file_options
- gnuplot_pm3d_options
- gnuplot_ps_term_command
- gnuplot_default_term_command

5.1 plot2d

plot2d is the CGN function for the Maxima plot2d. This can plot scatter plots and lisp functions (and lambda expressions too) . Try (cgn::plot2d-test) to see an example.

5.2 plot3d

plot3d is the CGN function for the Maxima plot3d. This can only plot functions. Try (cgn::plot3d-test) to see an example.

6 Using multiple connections

Since CGN 009, you can use more than one gnuplot connection at time. To do that, you must understand something about how CGN works.

CGN uses a special variable named `*gnuplot*` to maintain a hash table where it looks for open connections to gnuplot. Each connection is named by a symbol. When you call `start-gnuplot` (or `with-gnuplot` does this for you), CGN creates a connection named `'default-gnuplot-connection`. When you use some method from CGN, it looks for the name of the connection to use by looking at the variable `*default-connection-name*`⁶. Then, if you want to use more than one connection, just:

1. Create a new connection.
2. Bind `*default-connection-name*` to the name of the connection you want to use.
3. Just call the functions as you do usually. Everything will be sent to that connection.

You can create a new connection calling *add-new-gnuplot-connection name*. You can also close manually a connection calling *delete-gnuplot-connection name*.

6.1 The with-gnuplot-connection macro

The procedure of binding⁷ `*default-connection-name*` and plotting something is very usual. CGN provides a macro to do that. This is the `with-gnuplot-connection` macro. To do something with a connection, just use:

```
(with-gnuplot-connection name
  ;Put your code here
)
```

⁶Some methods understand the name of the connection as a parameter, too. Look above for the list of parameters that `plot2d` and `plot3d` understand.

⁷Please, you should never destructively set this variable. This would cause strange things to happen.

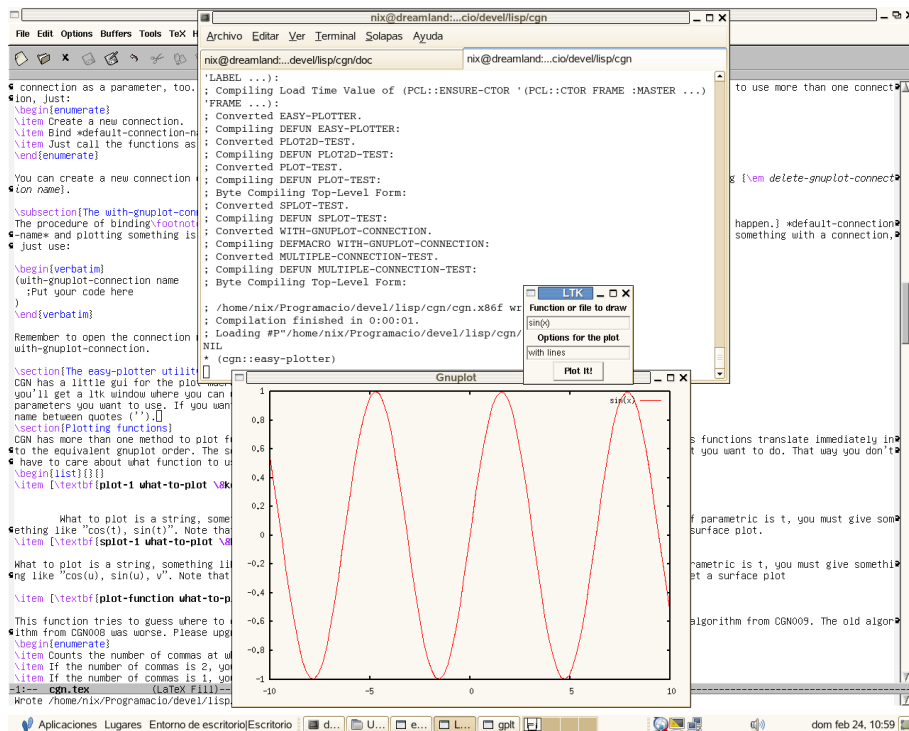


Figure 1: This is the new easy-plotter utility.

Remember to open the connection manually before calling with-gnuplot-connection.

7 The easy-plotter utility

CGN has a little gui for the plot macro. If you write (cgn::easy-plotter), you'll get a ltk window where you can write the function to plot and the parameters you want to use. If you want to plot a file, you must write It's name between quotes (""). See 1.

8 Plotting functions

CGN has more than one method to plot functions. The gnuplot-like method is to use the plot-1 and splot-1 functions. This functions translate immediately into the equivalent gnuplot order. The second method is to use the plot-function method. This function tries to guess what you want to do. That way you don't have to care about what function to use.

plot-1 what-to-plot &key parametric surface What to plot is a string, something like "cos(x)" or "x * tan(x)". Note that the variable MUST be called x. If parametric is t, you must give something like "cos(t), sin(t)". Note that in this other case, the variable must be called t. If surface is t, you get a surface plot.

splot-1 what-to-plot &key parametric surface What to plot is a string, something like "cos(x*y)" or "x - y". Note that the variables MUST be called x and y. If parametric is t, you must give something like "cos(u), sin(u), v". Note that in this other case, the variables must be named u and v. If surface is t, you get a surface plot

plot-function what-to-plot This function tries to guess where to call plot or splot. Its algorithm is easy to understand⁸:

1. Counts the number of commas at what-to-plot.
2. If the number of commas is 2, you are trying to do a 3D parametric plot (something like "splot u, u - v , v").
3. If the number of commas is 1, you are trying to do a 2D parametric plot (something like "plot cos(t), sin(t)").
4. If the number of commas is zero, you are trying to do a non parametric plot. Then It looks for a "y" letter in what-to-plot. If It finds It, tries to do an f(x,y) plot (like "splot x*y"). It calls (plot what-to-plot) in any other case.

plot-function works very well, but you always can use plot and splot if the algorithm doesn't understand what you want. Other functions in the library, like *animate-function* use plot-function to do Its work.

Note that you always must give an adequate name to your variables: t for a 2D parametric plot, u and v for a 3D parametric plot, etc... See figures 2, 3, 4 and 5 for examples. Also note that plot-function never does a surface plotting by itself. If you want to do that, enable surface plotting by (*enable-surface*), plot something with plot-function and disable surfaces with (*disable-surface*).

replot-function what-to-plot This is the same as plot-function, but using replot. Use this to add a second plot to the same screen. See figure 6

⁸This is the new algorithm from CGN009. The old algorithm from CGN008 was worse. Please upgrade to CGN009 if you have some problem with plot-function

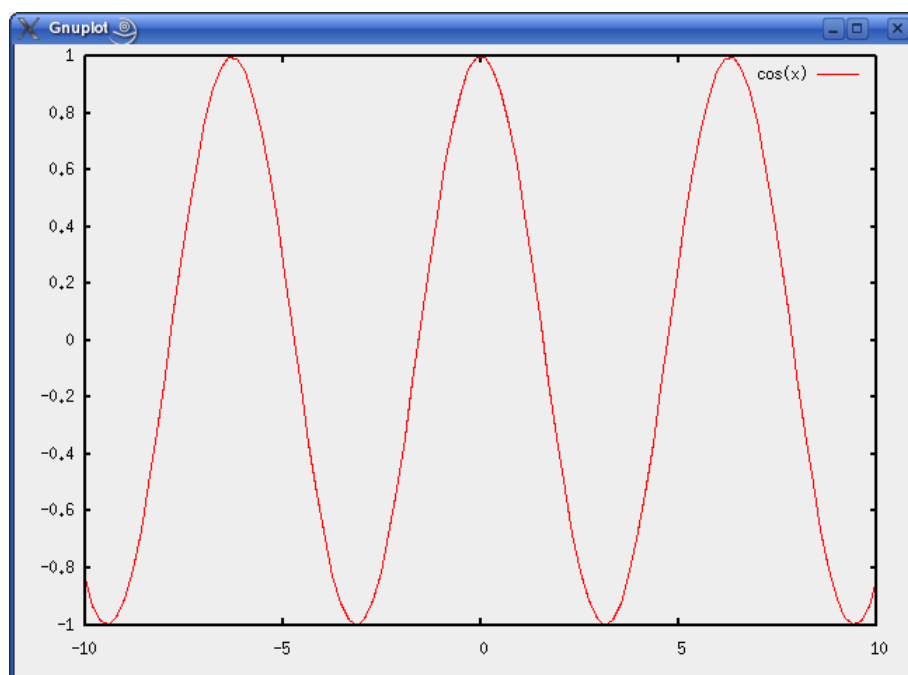


Figure 2: This can be the output from `(plot-1 "cos(x)" )` or `(plot-function "cos(x)" )`

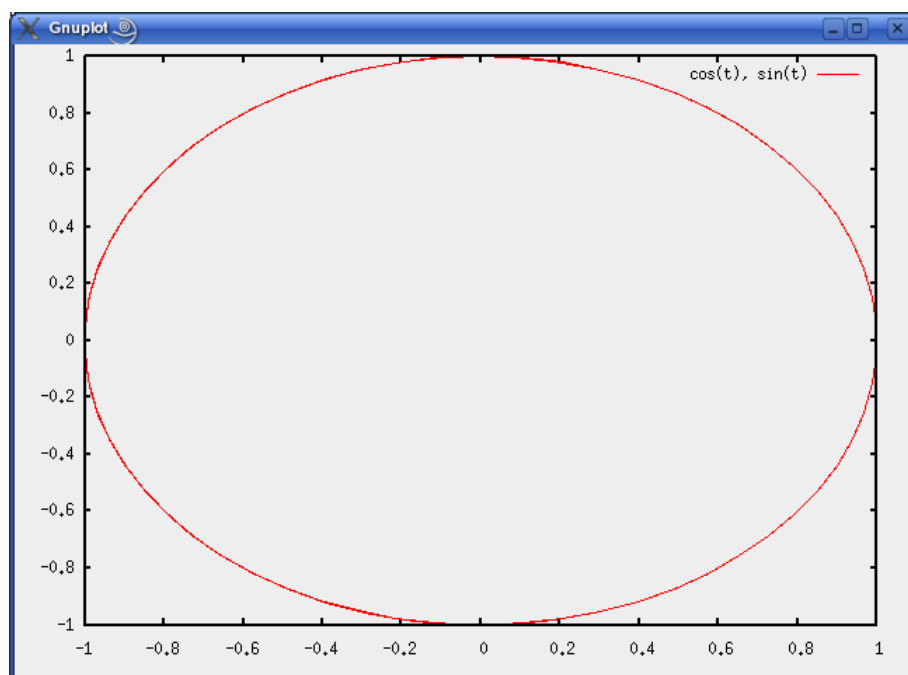


Figure 3: This is a circle. Can be the output from (plot-1 "cos(t), sin(t)" :parametric t) or (plot-function "cos(t), sin(t)")

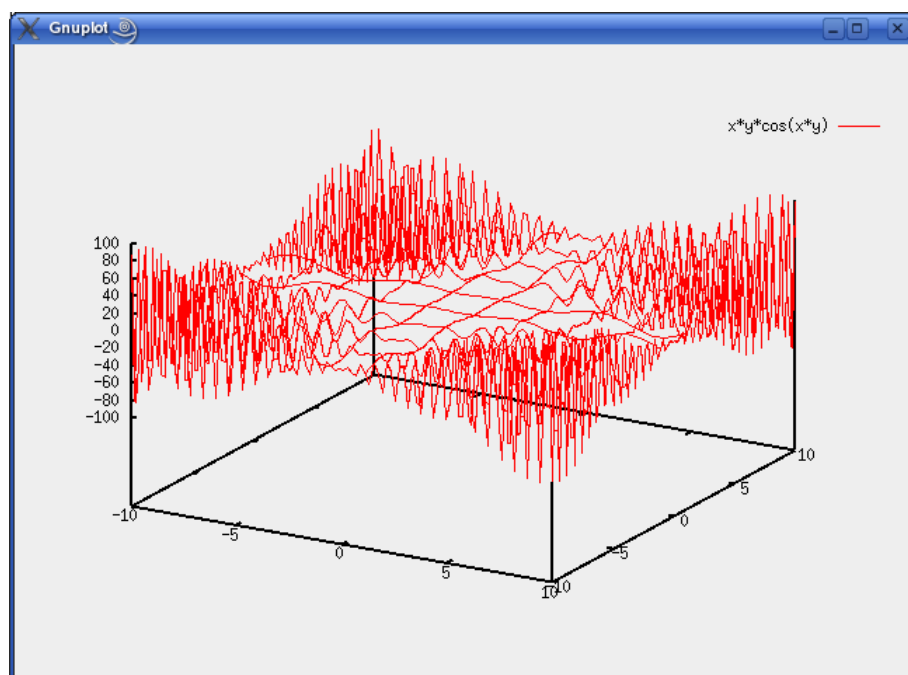


Figure 4: The output from `(splot-1 "x*y*cos(x*y)")` or `(plot-function "x*y*cos(x*y)")`

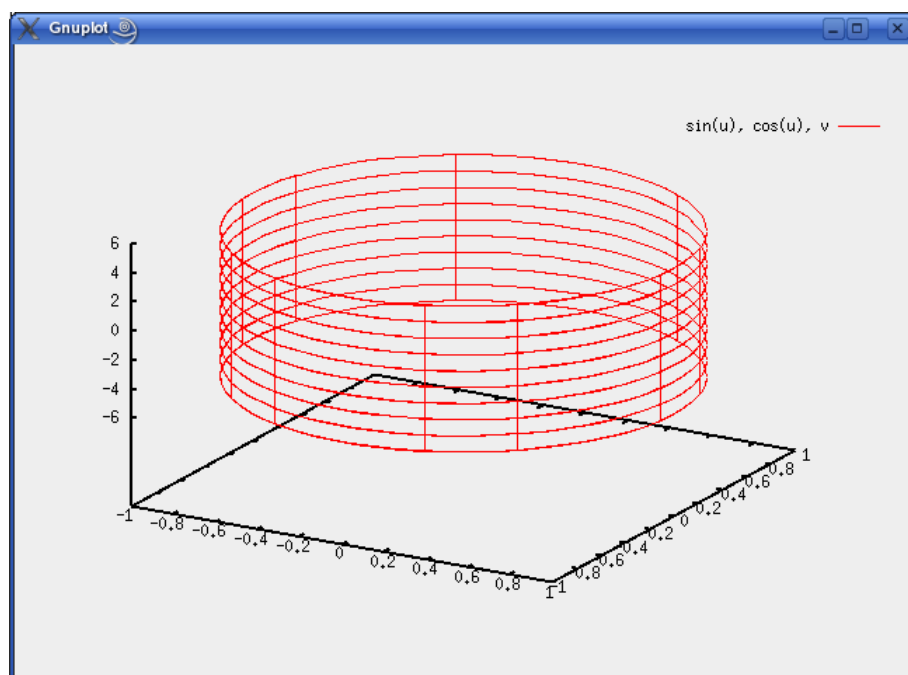


Figure 5: A beautiful cilinder. You can draw It with (splot-1 "sin(u), cos(u)
, v " :parametric t) or (plot-function "sin(u), cos(u) , v ")

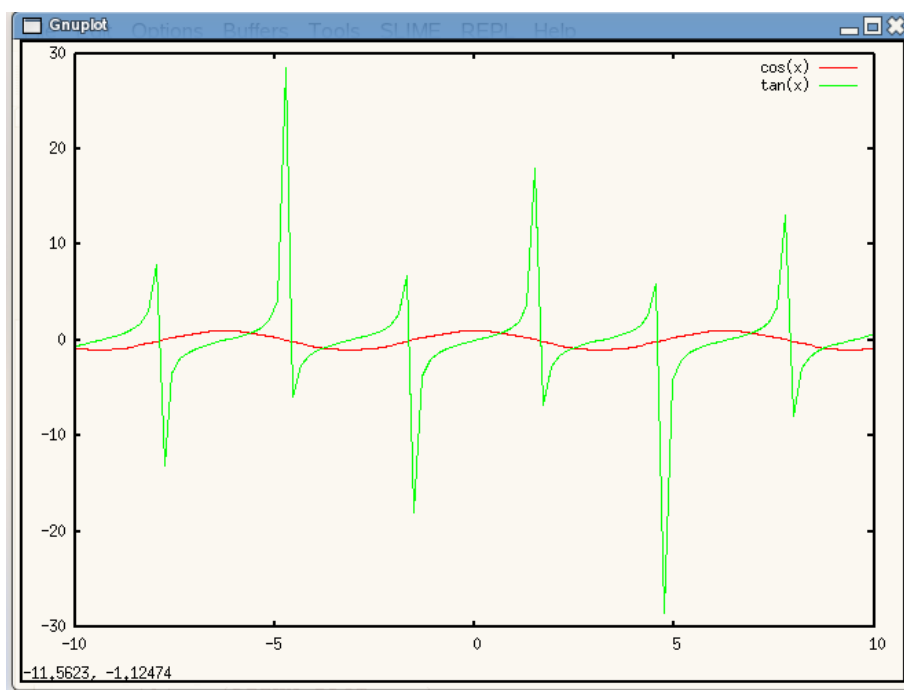


Figure 6: Two function plots at the same screen. This was generated with
(with-gnuplot () (plot-function "cos(x)") (replot-function "tan(x)"))

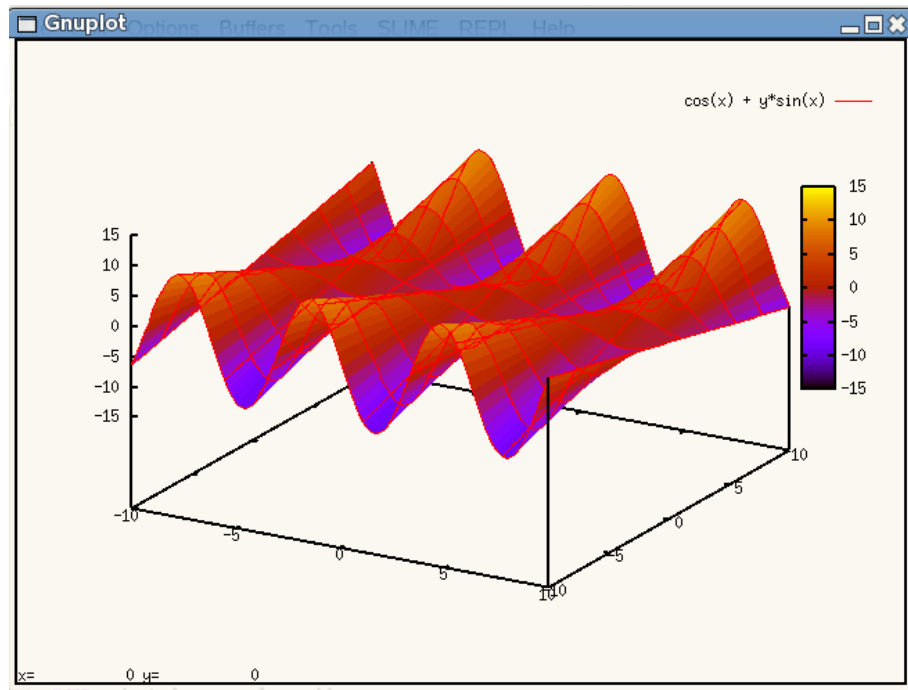


Figure 7: A very simple surface plot. This was generated with (with-gnuplot
 () (splot-1 "cos(x) + y*sin(x)" :surface t))

8.1 Surfaces

CGN uses the pm3d module to plot surfaces. You've seen above that you can set :parametric t when you call splot or plot to plot a surface. You can also use enable-surface and disable-surface to manage this manually.

enable-surface Simply sets the pm3d module.

disable-surface Unsets the pm3d module.

See figure 7 for an example.

9 Plotting scatter graphics

CGN can create scatter graphics easily. To do that, first of all you must create a scatter-graphic description for your graphic. Then you can add data series to it. Finally, use plot-scatter-graphic to plot the scatter-graphic description.

create-scatter-graphic-description title This creates a description object for your scatter graphic. Once this is created, you have to add data

series to It.

create-scatter-data-serie (sc scatter-graphic) x y x_error y_error title This is the method used to add data series to a description object. x, y, x_error and y_error are lists. Title is the title for the data serie. Note that all the list need to have the same length. If you don't want error in the x or y axis, just set x_error or y_error to a list filled with zeros, with the adequate length.

plot-scatter-graphic scatter-graphic &key fit-function This function will plot your scatter graphic description. fit-function is a function (a gnuplot string) which is supposed to match you points . If you don't provide It, you'll get a normal scatter plot.

Example: create an scatter graphic with points from experiment1: (1,2) , (2,3), (3,4) without errorbars and points from experiment2: (2.5,1) , (3.1,2), (4.5, 7) ,(8,9) with a 10% of error for x and y.

```
(setq sc (create-scatter-graphic-description "Our experiments"))

(create-scatter-data-serie sc '(1 2 3) '(2 3 4) '(0 0 0)
'(0 0 0) "Experiment1")

(create-scatter-data-serie sc '(2.5 3.1 4.5 8) '(1 2 7 9)
'(0.25 0.31 0.45 0.8) '(0.1 0.2 0.7 0.9) "Experiment2")

(plot-scatter-graphic sc)
```

See figure 8 to see the output.

9.1 The simple way

plot-points x y &key x_error y_error This is easier to use if you want a scatter graphic with only 1 data series on It. You provide lists to this function, all them with the same length. If you provide x_error, you'll get errorbars at the x axis. If you provide y_error, you'll get errorbars at the y axis. All the combinations of x_error and y_error are possible.

Example:

```
(with-gnuplot ()
  (plot-points
    (list 1 2 3 4 5)
    (list 2 3 4 5 6 )
```

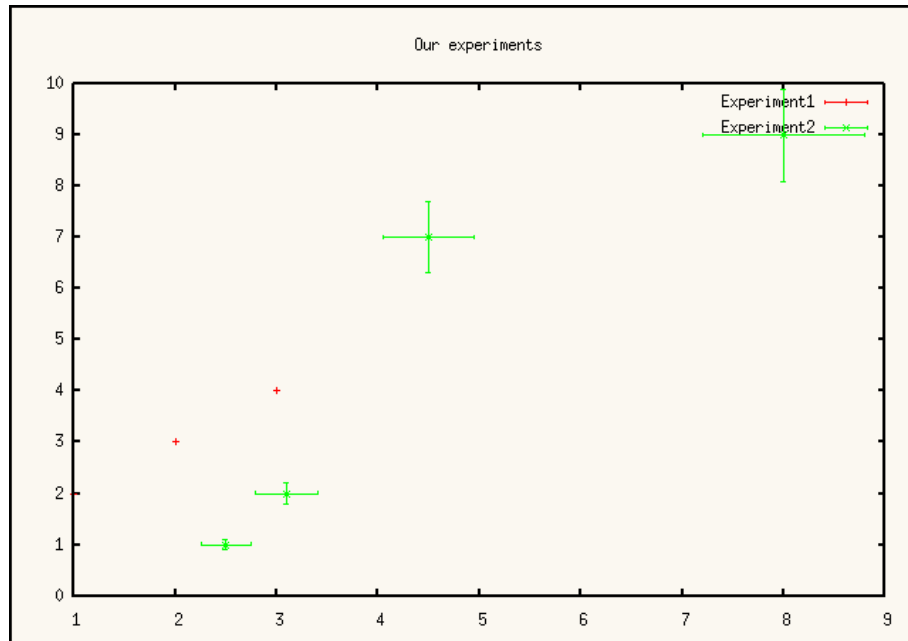


Figure 8: A scatter graphic

```

        :y_error (list 0.1 0.05 0.12 0.07 0.1 )
    )
)

```

See figure 9.

9.2 plot-points-and-function

This has been contributed by Dan Becker. The prototype of this function is :

plot-points-and-function (x y func xstart xend data-title func-title)

This function is used to plot a x-y data serie plus a function which is supposed to match. x and y are sequences (lists or vectors) with the same length, func is a string containing a gnuplot function, xstart and xend is the range to use in the x axis , data-title and func-title are the titles to show in the graphic for the data series and the function. Please run (cgn::plot-points-and-function-test) to see a little example.

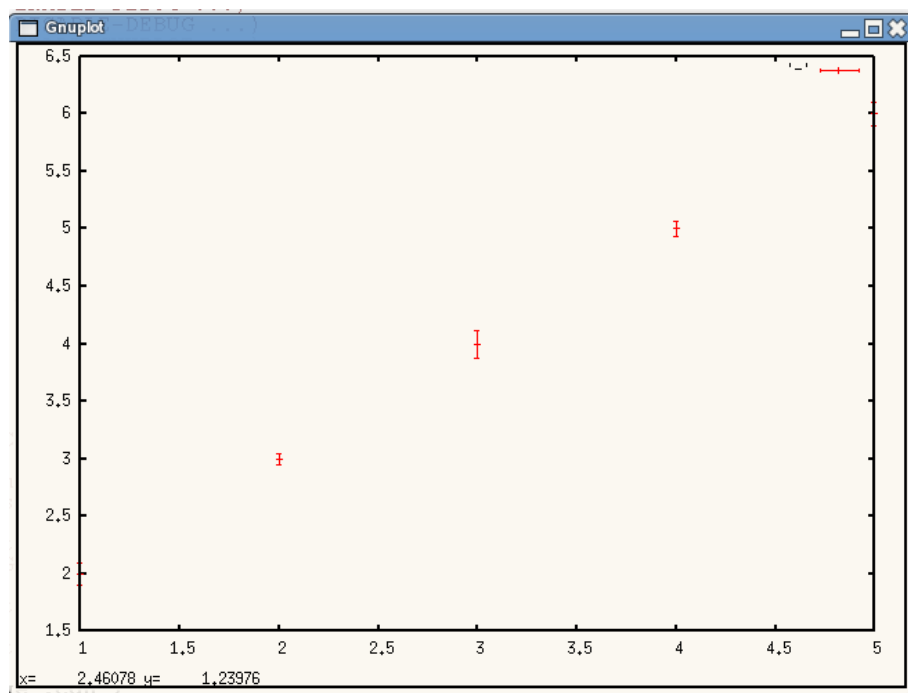


Figure 9: A scatter graphic made with plot-points

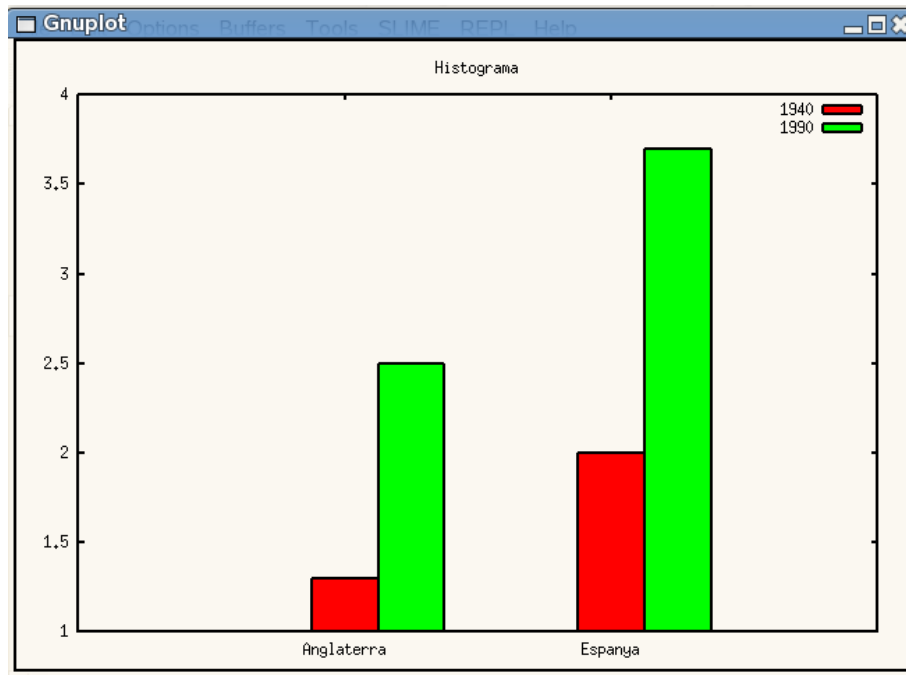


Figure 10: A more elaborated histogram, with two data series.

10 Plotting histograms

CGN can plot histograms. The `plot-histogram` function makes this easy. Note that this function has a side effect: it resets gnuplot after showing the plot. **plot-histogram** *xtics serie* *xtics* is a list with the titles appearing at the xaxis, under each column. *serie* is the height of each column.

With `plot-histogram` you get a very simple histogram. You can do more complicated histograms with `plot-complex-histogram`. See figure 10

```
(with-gnuplot ()
  (let
    (
      (hg (create-histogram-description "Histograma" (list "Anglaterra" "Espanya" ) )
    )
    (create-histogram-data-serie hg (list 1.3 2 ) "1940" )
    (create-histogram-data-serie hg (list 2.5 3.7 ) "1990" )
    (plot-complex-histogram hg )
  )
)
```

11 Configuring gnuplot

CGN can do some personalization of gnuplot for you. At the moment:

```
set-grid      on-off
  set-range    eix min max
  set-title    text
```

12 Printing

CGN can print directly from gnuplot. To do that, It needs to know what operative system is running⁹. Just use:

```
print-graphic  &optional (os *os*)
```

os must be set to 'w32'¹⁰ on windows. For non-windows machines, you can use whatever symbol except 'w32': linux, unix, my-beautiful-machine, etc...

13 Saving/loading

CGN can save a gnuplot session to a file, and It can load It later. You have to understand that CGN does not save nothing. It's gnuplot who saves the session. Then, when you load the saved session, you can restore gnuplot's status (plotted functions, title, etc...), but not CGN's status.

To save use:

```
save-cgn      file
```

And to load:

```
load-cgn      file
```

⁹This is only until release 007. Release 008 can determinate which os is running without the help of the user.

¹⁰There's an easy trick to conditionally set *os* to the correct symbol. Just (if (member :win32 *features*) (setq *os* 'w32) (setq *os* 'another)). This code was written for CGN 008, but It's not included in It because I don't know If works everywhere. I've tried It with sbcl and clisp and works. Please, if you've tried It with another lisp for windows, send me a mail to superratoli@gmail.com and this code will be included in CGN 009.

14 Creating animations

Since CGN 008, CGN can create animations. To do that, simply use `animate-function`.

`animate-function` *what-function* *&key* *what-parameter* *initial-value* *increment* *number-of-frames* *fps*

what-function simply is a string, where you use some parameter. CGN will change the value of that parameter. Example: `"sin(a*x)"`. CGN will change the value of *a*.

what-parameter is the variable that CGN will change. In our example, the string `"a"`.

initial-value is a number, the initial value for *a*.

increment is the number to add (this can be a negative number) to the parameter to compute the next frame.

number-of-frames is the total number of images in the animation.

fps (frames per second) is the number of frames per second that CGN will show.

Here you have a little animation. Copy It to your REPL and see what It does.

```
(with-gnuplot ()
  (animate-function "cos(a * x ) + a * sin(x) "
    :variable "a"
    :initial_value 0.0
    :increment 0.01
    :number_of_frames 100
    :fps 10 ) )
```

15 Sending commands directly to gnuplot

CGN simply sends commands to gnuplot writing to a pipe. You can do that too. This way, you can do everything that gnuplot can do, even if CGN doesn't have implemented It. Simply use `format-gnuplot`.

`format-gnuplot` *text* *&rest arguments* `format-gnuplot` works exactly like the habitual `format` function in lisp, but It doesn't have a stream parameter. This is just the pipe. Example: suppose you want to plot the function

`cos(x)`. CGN can do that, but this is an example. In gnuplot, you would write `"plot cos(x)"`. Well, you just could use `(format-gnuplot "plot cos(x)" )`. Play a little with this function and you'll see how easy is to use It.

16 Debugging CGN

Since CGN 008, debugging is supported. You shouldn't enable debug for a normal use. Enable It only if you are developing some kind of extension to CGN. To enable debug, just use:

```
(enable-debug)
```

And to disable It:

```
(disable-debug)
```

With debug enabled, CGN will print to `*standard-output*` everything that It sends to gnuplot.

17 Inside CGN

Basically, CGN is constructed over two different parts:

- A pipe that can communicate with gnuplot and a function to send messages to It and
- Lots of Lisp functions wrapping the most common operations you can do with gnuplot.

CGN uses Ltk for the pipe issue. Ltk is a graphical toolkit for Lisp that uses a pipe to communicate with Tk¹¹. It runs everywhere a Lisp implementation exists and is well supported and maintained. It provides a function that creates a pipe and returns a stream to the executed program. But with 2 goodies: It's maintained by other people than me (less work for me) and wraps code for lots of different Lisp implementations. Using It, CGN works everywhere.

¹¹Tk is the Tcl ToolKit. It's an interpreter that, when called without arguments, waits for code on standard input. Ltk just creates the Tk code needed to create windows, buttons, etc...

17.1 Extending CGN

It's easy to extend CGN. Everything you have to do is create a new function that sends to gnuplot the code that fills your needs, using `format-gnuplot`. `format-gnuplot` is just like `format`, but provides some features like debuggind. It also makes sure that gnuplot executes the command you send.

The first thing you have to do is creating a package for putting your code in It:

```
(defpackage :extending-cgn-example
  (:use :cl :cl-user :cgn)
  (:export #:set-property))
```

```
(in-package :extending-cgn-example)
```

To extend CGN It's not necessary create classes. You just have to write functions that send the code that fills your needs. At this example, we face a very common problem: there are lots of flags that control the behaviour of gnuplot. CGN doesn't have functions to set all of them. Well, now we're gonna write a function to set all the flags that don't need paramethers.

First of all, we study what the generated code will seem:

```
set timestamp
set arrow
...
```

It becomes clear that this function will do the work:

```
(defun set-property ( property )
  (format-gnuplot "set ~A" property ))
```

Now we can test this function and see what It happens:

```
* (start-gnuplot)
* (enable-debug)
T
* (plot-function "cos(x)")
plot cos(x)
* (set-property "timestamp")
set timestamp
* (plot-function "sin(x)")
plot sin(x)
```

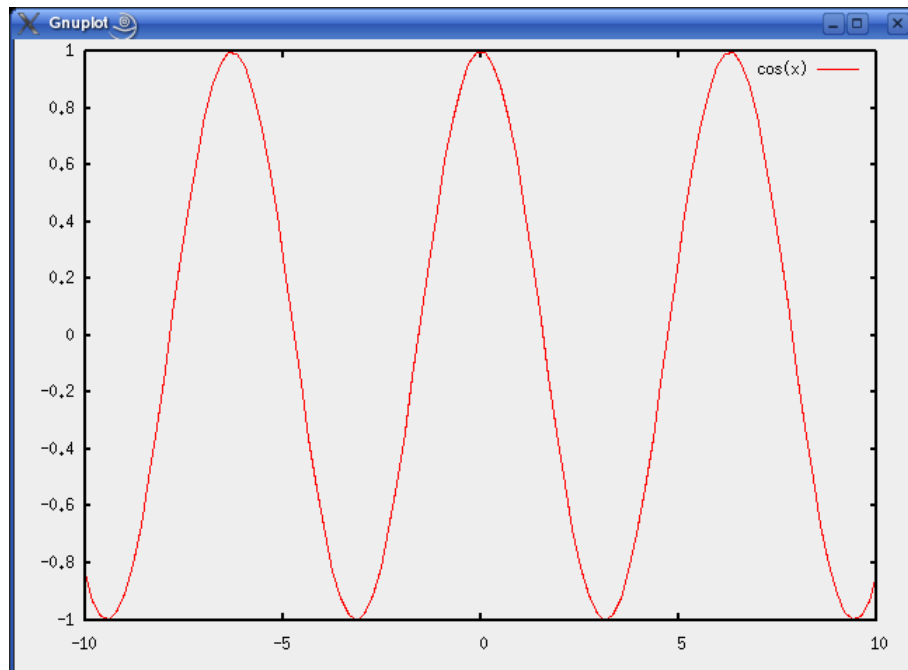


Figure 11: Before calling set-property

18 Seeing release info

Just type

```
(show-release-info)
```

And you'll see something like this:

CGN 008 released 9-II-07

19 Changelog

CGN 009

Now CGN never uses tmp files. This prevents some extrange bugs from CGN008.

9-II-07: CGN 008

Added plot-scatter

Added create-scatter-data-serie

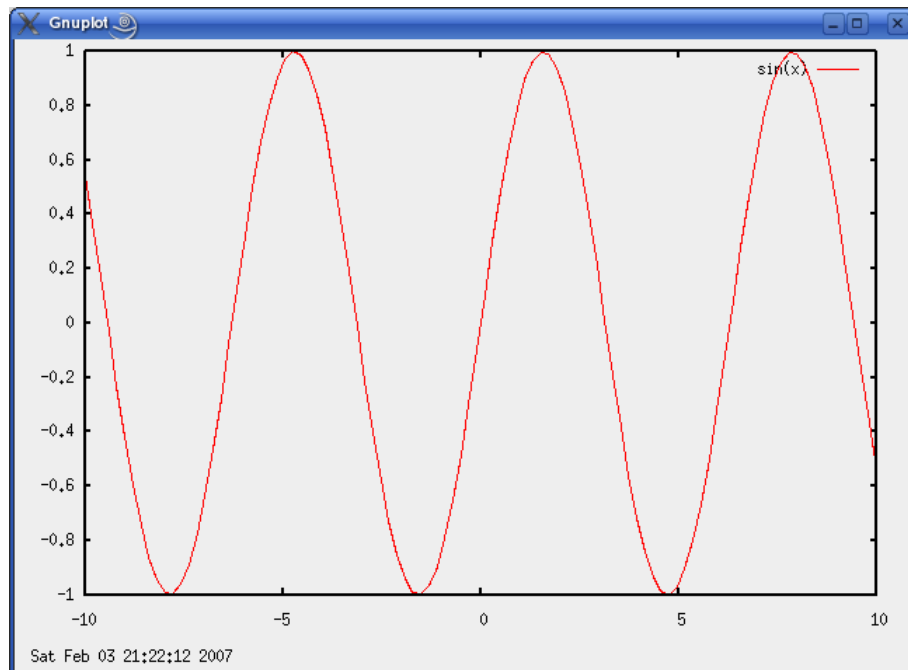


Figure 12: After calling set-property. Watch the timestamp.

Added create-scatter-graphic-description

Added new structures : data-serie and scatter-graphic, to save information about scatter graphics.

Added show-release-info.

Added two new flags: *release* and *release-date*.

format-gnuplot now does debug if debugging is enabled.

Added enable-debug and disable-debug.

Added *debug*, a flag to control if CGN must do debug.

All the library has been updated to use the new format-gnuplot facilities.

Format-gnuplot now understands all the options that format can understand.

Replot-function now can replot every type of functions.

Created a new manual for CGN.

Animate-function now can create animations for whatever type of function, only for one parameter.

Added animate-function, to create animations. At the moment, only for $f(x)$, and one parameter.

plot-function now plots using the parametric form.

20-VI-06: CGN 007

Added print-graphic, to print directly from CGN.

Modified plot-function. Now can be used to plot $g(x,y)$, not only $f(x)$.

Added save-cgn, to save.

Added load-cgn, to load.

Added with-gnuplot, a macro which should be the preferred way to use CGN.

Added the `*os*` special variable, to use with with-gnuplot to let CGN know which os is running. This is used to print, at the moment.

Updated cldoc documentation.

8-VI-06: CGN 006

Corrected bug. Gnuplot kepted running later of using close-gnuplot.

Comments now translated to English. It seems that the original Catalanian documentation produced slow compilations at non Catalanian machines.

Added cldoc documentation.

20 Bibliography

The oficial pdf manual of Gnuplot 4.2.2.

The gnuplot demos at the demo/ subdirectory of Its source tree.

I used the histogram.py program by Mustafa Paksoy to learn about xtics and histograms.

The oficial documentation of Maxima.

In the beginning, cgn was inspired by the gnuplot.py package. Google "gnuplot.py" to find It.

Lots of tutorials and examples from the www.